

# AmbientTalk: Object-oriented Event-driven programming in Mobile Ad hoc Networks

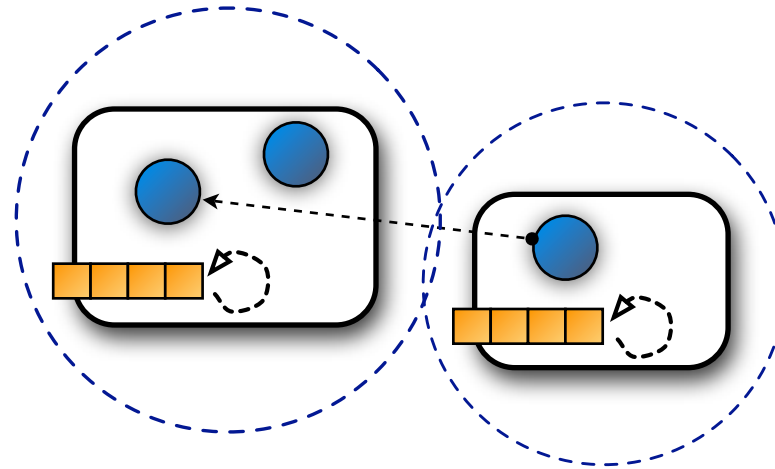
Tom Van Cutsem    Stijn Mostinckx    Elisa Gonzalez Boix  
Jessie Dedecker    Wolfgang De Meuter

Programming Technology Lab  
Vrije Universiteit Brussel  
Brussels, Belgium



# Context

Object-oriented programming languages



Software

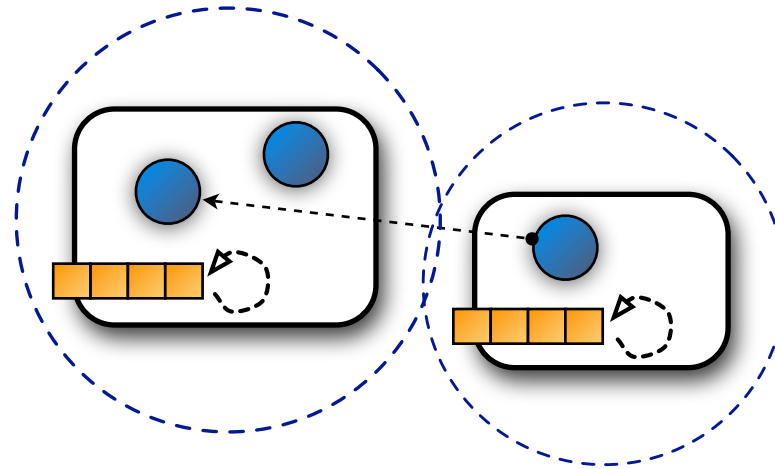
Hardware



Pervasive Computing  
(Mobile Networks)

# Context

Object-oriented programming languages



Software

Hardware



Pervasive Computing  
(Mobile Networks)

# Mobile Ad hoc Networks



# Mobile Ad hoc Networks



# Mobile Ad hoc Networks



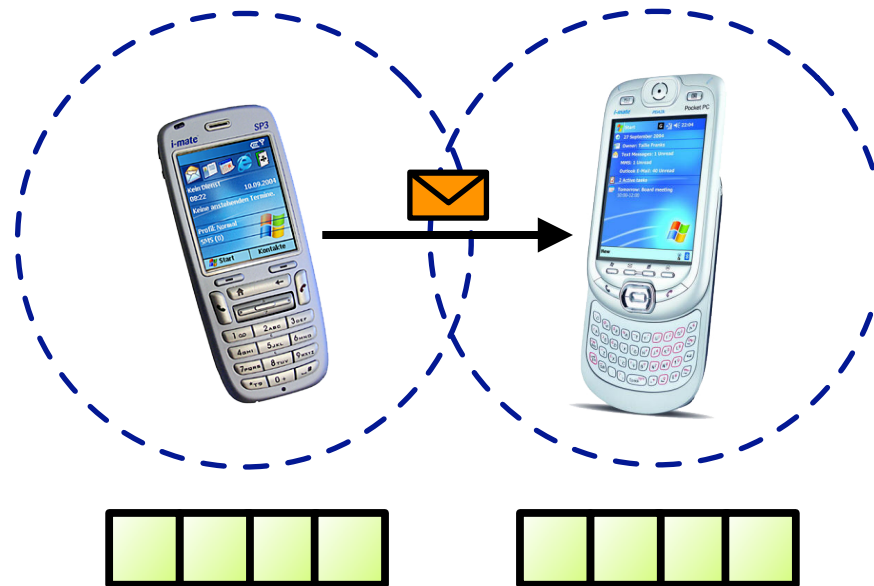
# Loose Coupling

Decoupling communication in *Time & Synchronisation*  
reduces impact of volatile connections



# Loose Coupling

Decoupling communication in *Time & Synchronisation*  
reduces impact of volatile connections





# Loose Coupling

Decoupling communication in *Time & Synchronisation*  
reduces impact of volatile connections

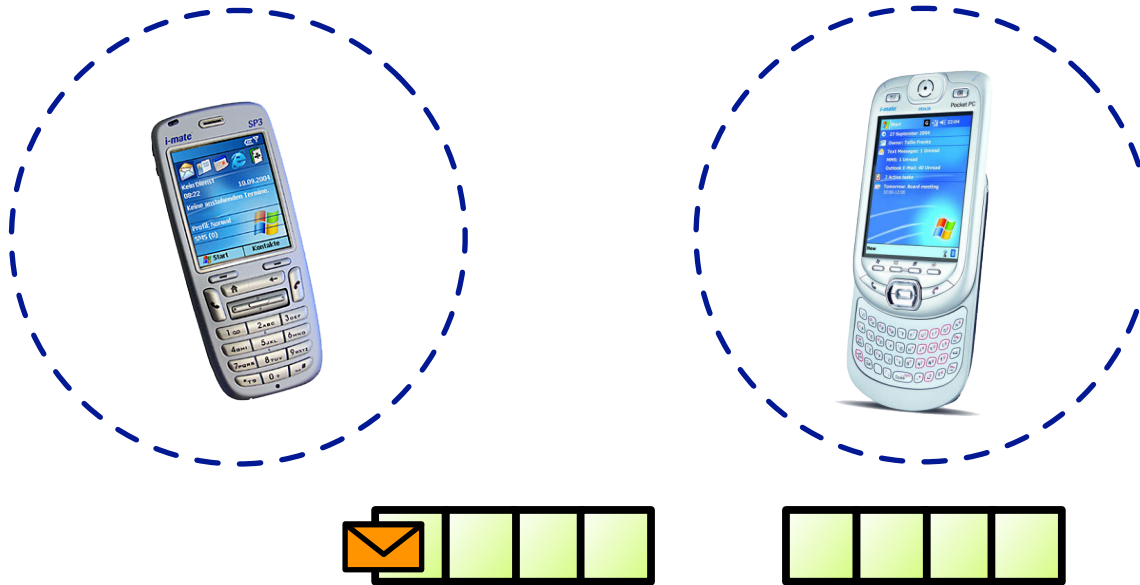


asynchronous  
send



# Loose Coupling

Decoupling communication in *Time & Synchronisation*  
reduces impact of volatile connections



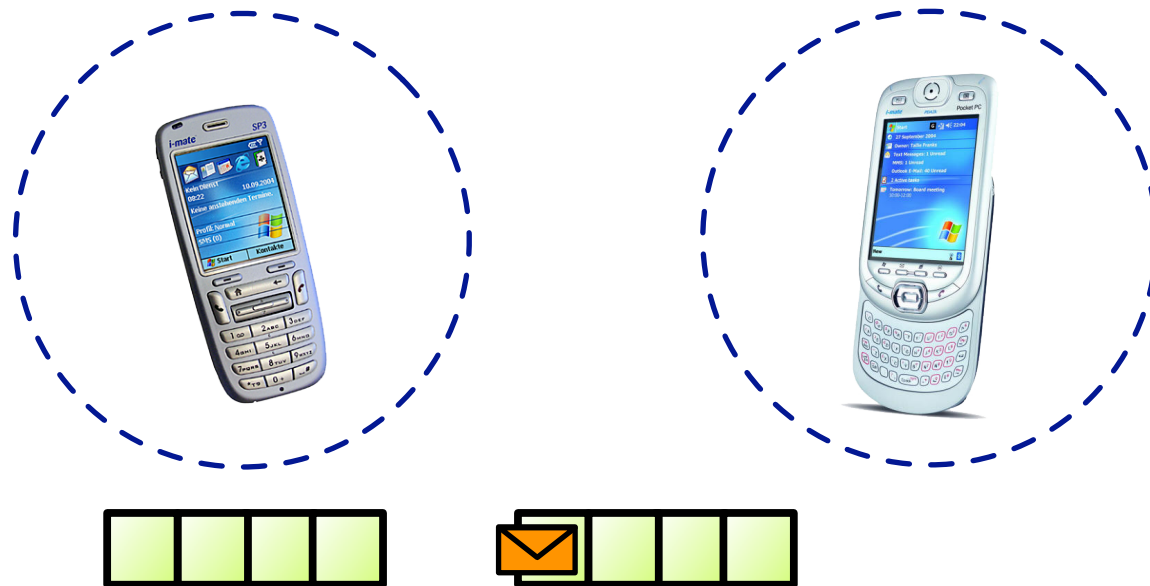
# Loose Coupling

Decoupling communication in *Time & Synchronisation*  
reduces impact of volatile connections



# Loose Coupling

Decoupling communication in *Time & Synchronisation*  
reduces impact of volatile connections



# Loose Coupling

Decoupling communication in *Time & Synchronisation*  
reduces impact of volatile connections



asynchronous  
receive

# Loose Coupling

Decoupling communication in *Space*  
enables ad hoc anonymous collaborations



# Loose Coupling

Decoupling communication in *Space*  
enables ad hoc anonymous collaborations



# Loose Coupling

Decoupling communication in *Space*  
enables ad hoc anonymous collaborations



provide service



# Loose Coupling

Decoupling communication in *Space*  
enables ad hoc anonymous collaborations



provide service

require service

# Ubiquitous Flea Market

Example: buy/sell concert tickets to proximate peers



# Ubiquitous Flea Market

Example: buy/sell concert tickets to proximate peers



# AmbientTalk: the language

- Distributed **object-oriented** language
- **Event-driven** concurrency based on **actors** [Agha86]
- **Future-type asynchronous** message sends
- Built-in **publish/subscribe** engine for service discovery of remote objects



# AmbientTalk: the project

- Started in 2005
- Small team: 3-6 people
- Interpreter
- Pure Java implementation
- Runs on J2ME/CDC phones



# Objects

```
def Item := object: {  
  def category;  
  def description;  
  def ownerContactInfo;  
  def init(c,d,o) {  
    category := c;  
    description := d;  
    ownerContactInfo := o;  
  }  
  def getContactInfo() {  
    ownerContactInfo  
  }  
  def placeSupply() {...}  
  def placeDemand() {...}  
}
```

- Prototype-based
- Objects are created:
  - anonymously
  - by cloning others

```
def ticket := Item.new(ConcertTicket,"...","...");  
ticket.placeDemand();
```

# Extensible language

```
def fac(n) {  
  (n = 0).ifTrue: {  
    1  
  } ifFalse: {  
    n * fac(n-1)  
  }  
}
```

- Block closures
- Keyworded messages
- Interfacing with JVM

# Extensible language

```
def fac(n) {  
  (n = 0).ifTrue: {  
    1  
  } ifFalse: {  
    n * fac(n-1)  
  }  
}
```

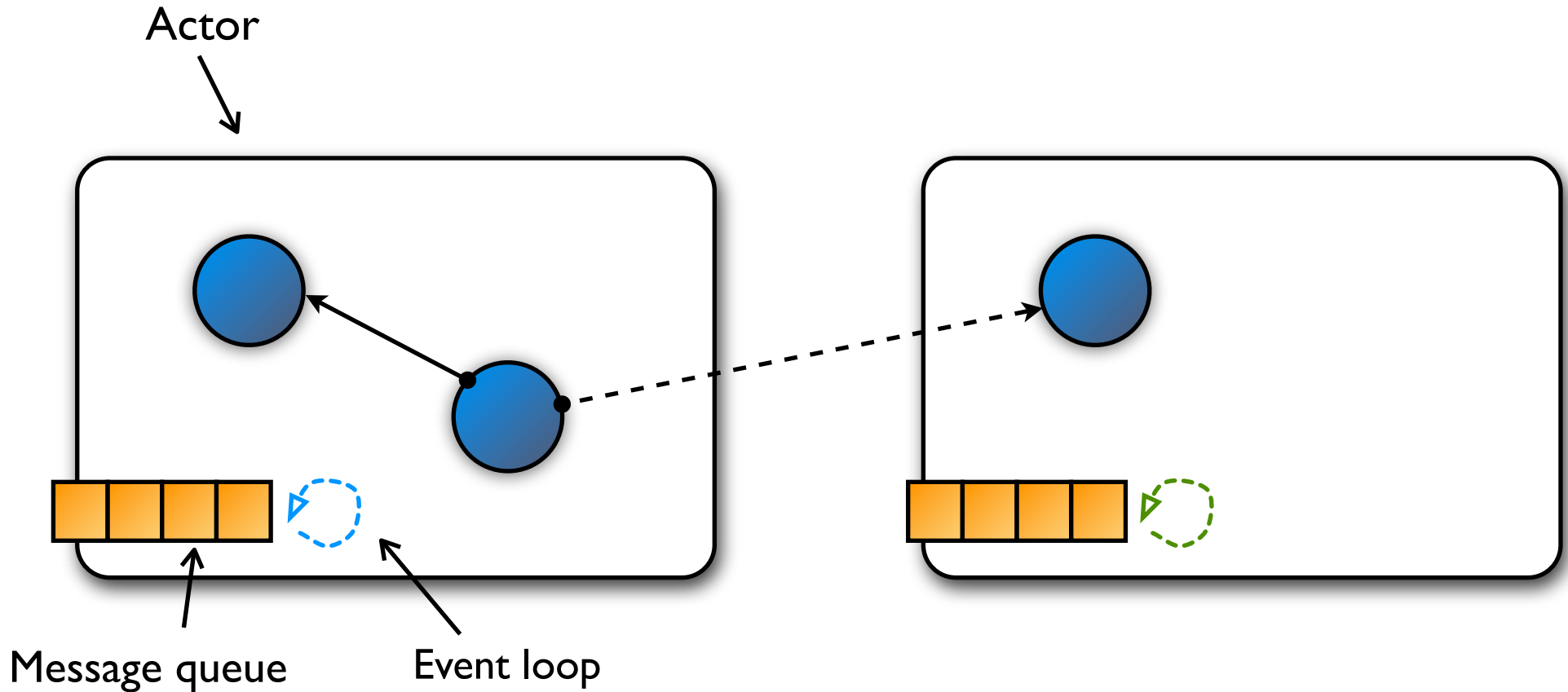
- Block closures
- Keyworded messages
- Interfacing with JVM

```
def Button := jlobby.java.awt.Button;  
def b := Button.new("test");  
b.addActionListener(object: {  
  def actionPerformed(ae) {  
    println("button pressed");  
  }  
});
```



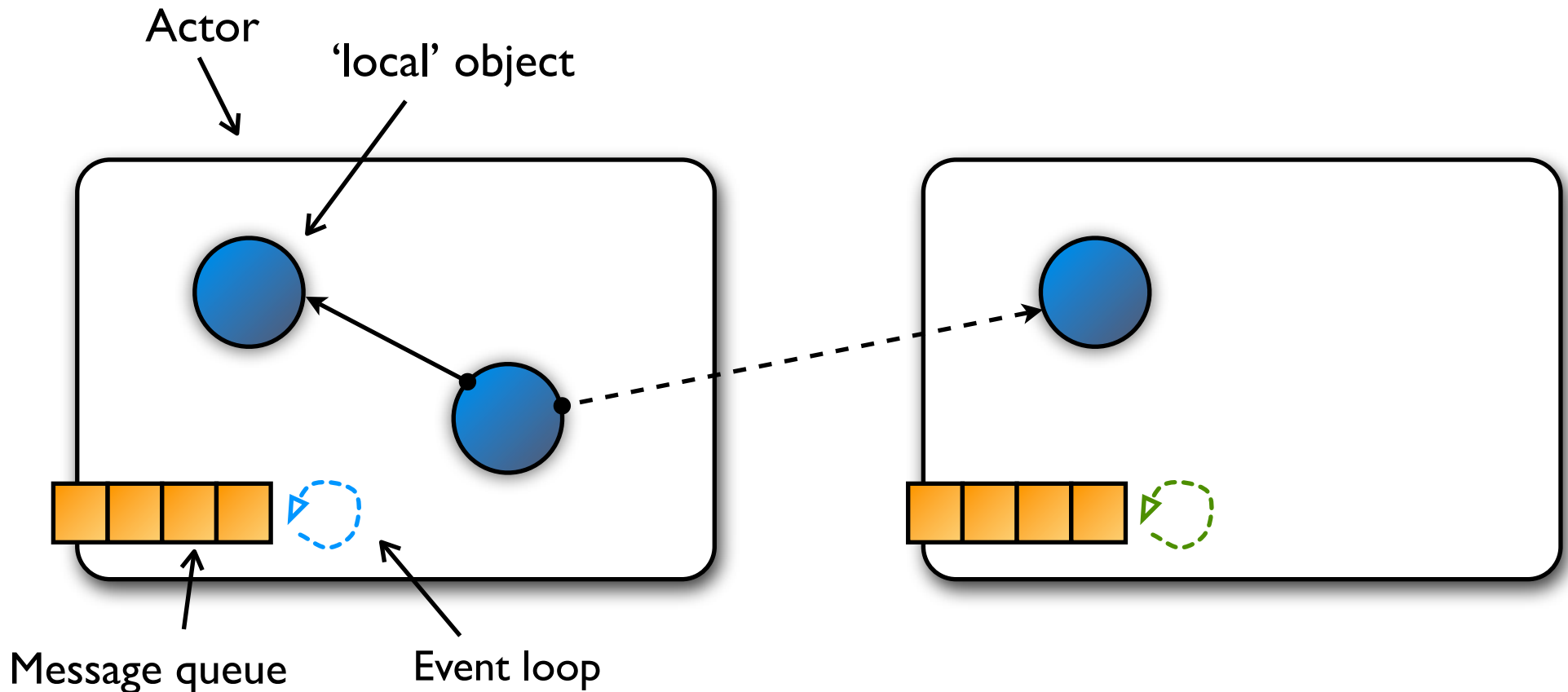
# Event loop concurrency

Based on E programming language [Miller05]



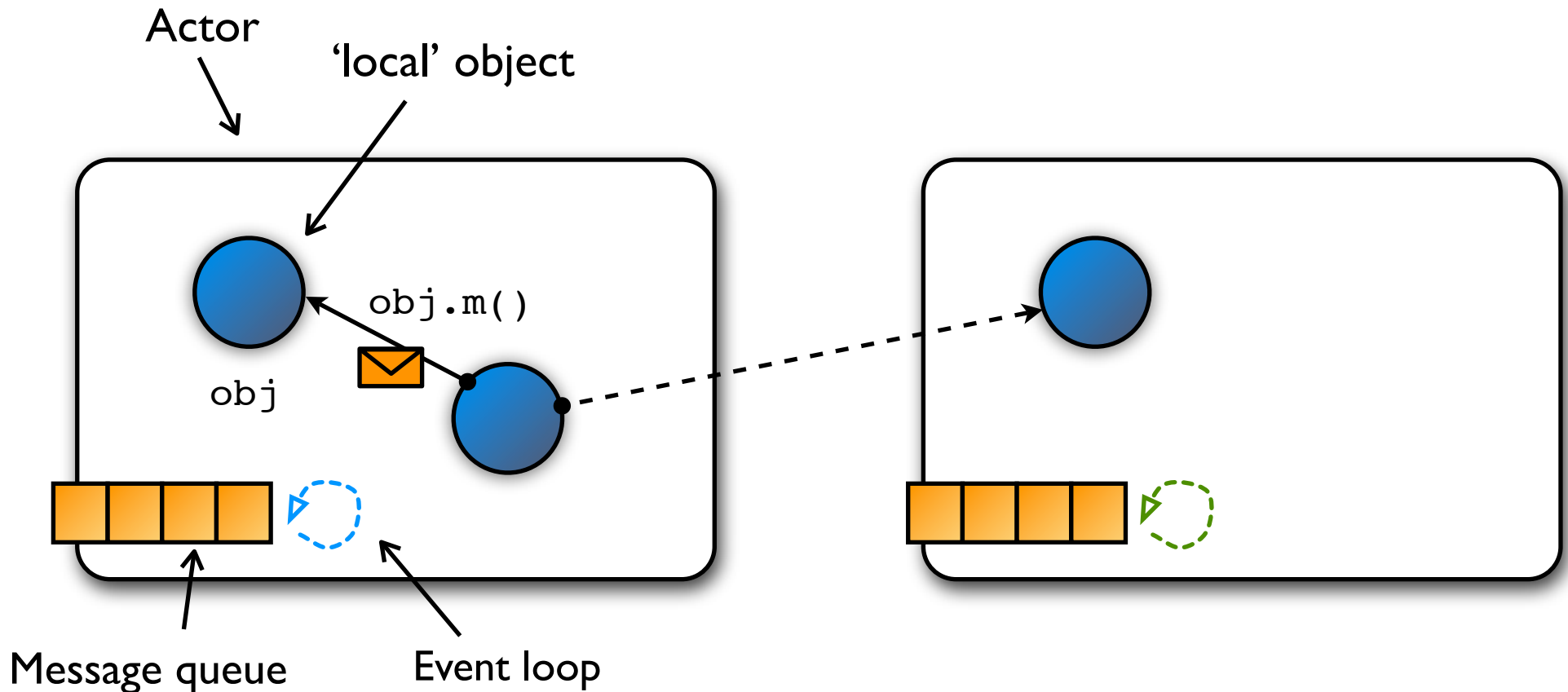
# Event loop concurrency

Based on E programming language [Miller05]



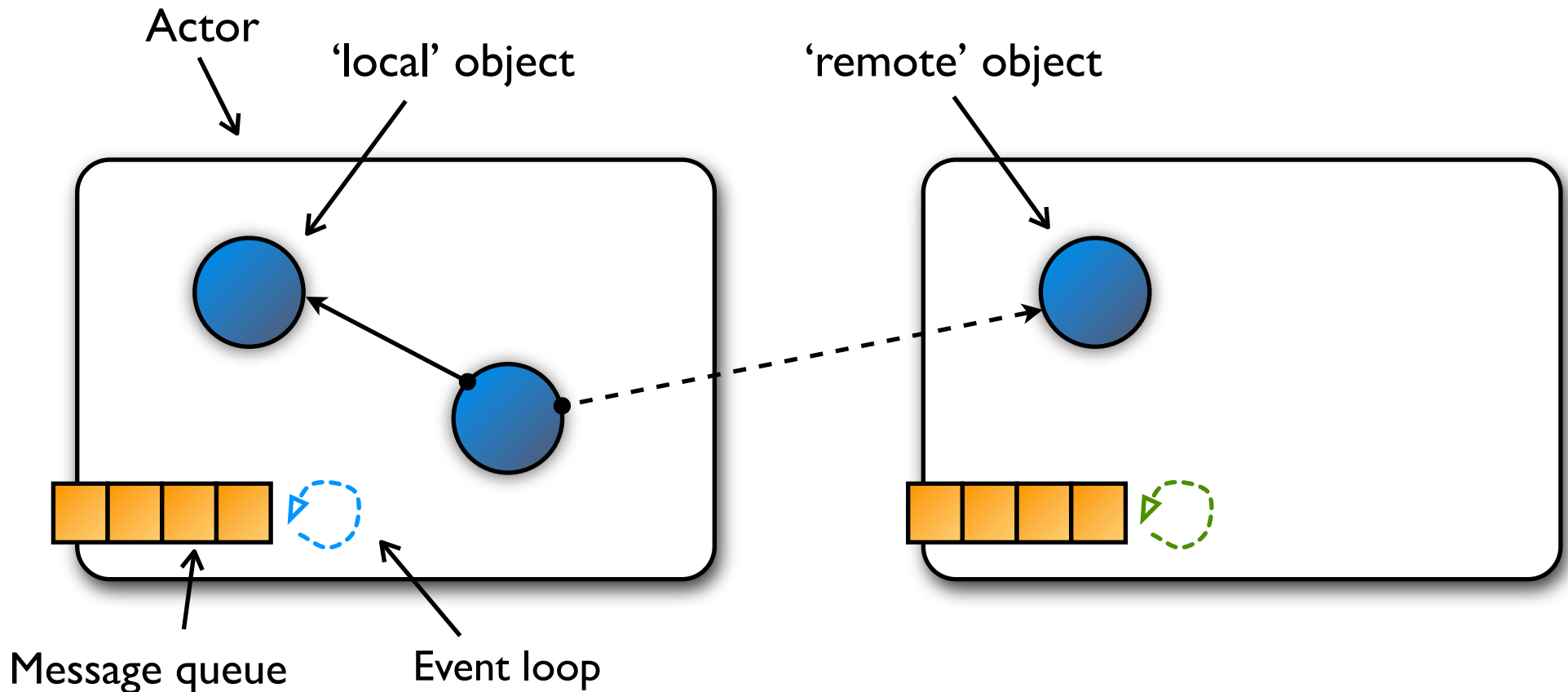
# Event loop concurrency

Based on E programming language [Miller05]



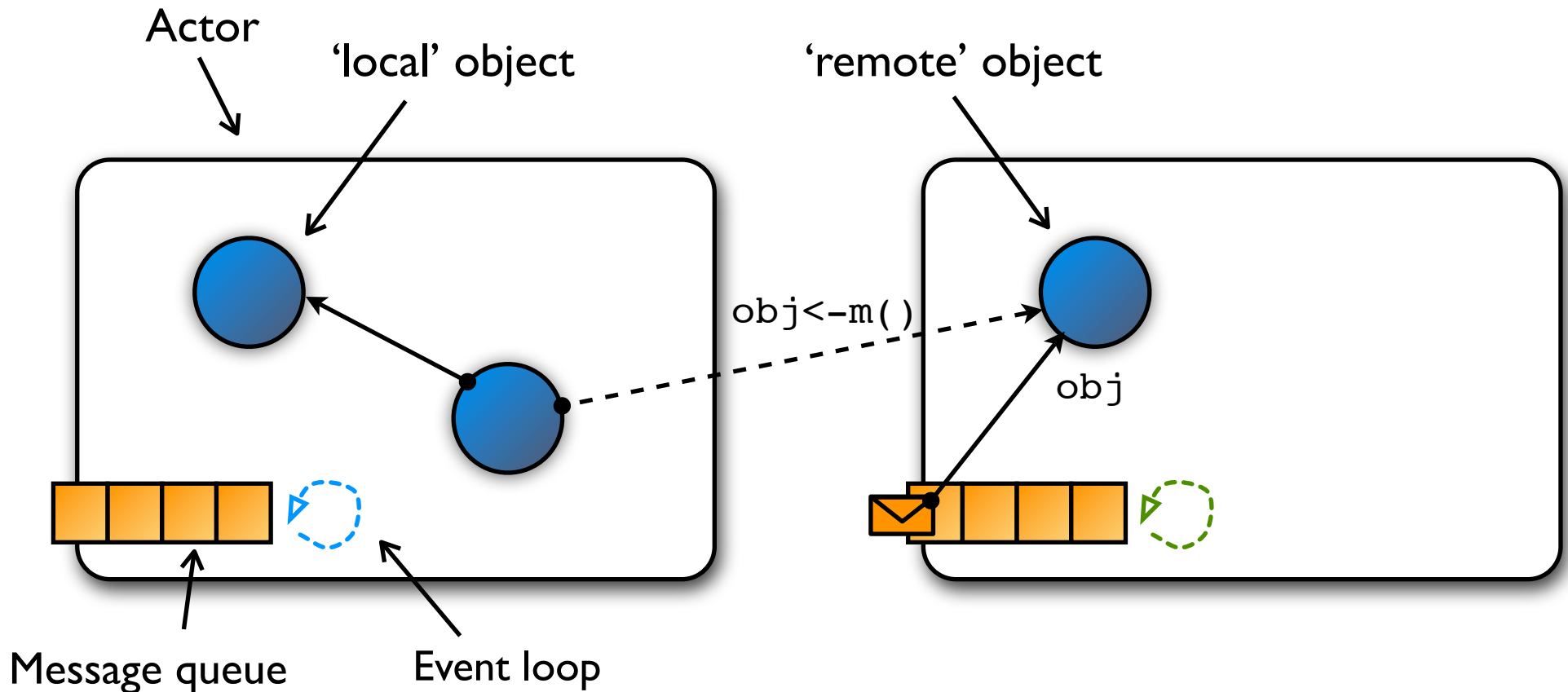
# Event loop concurrency

Based on E programming language [Miller05]



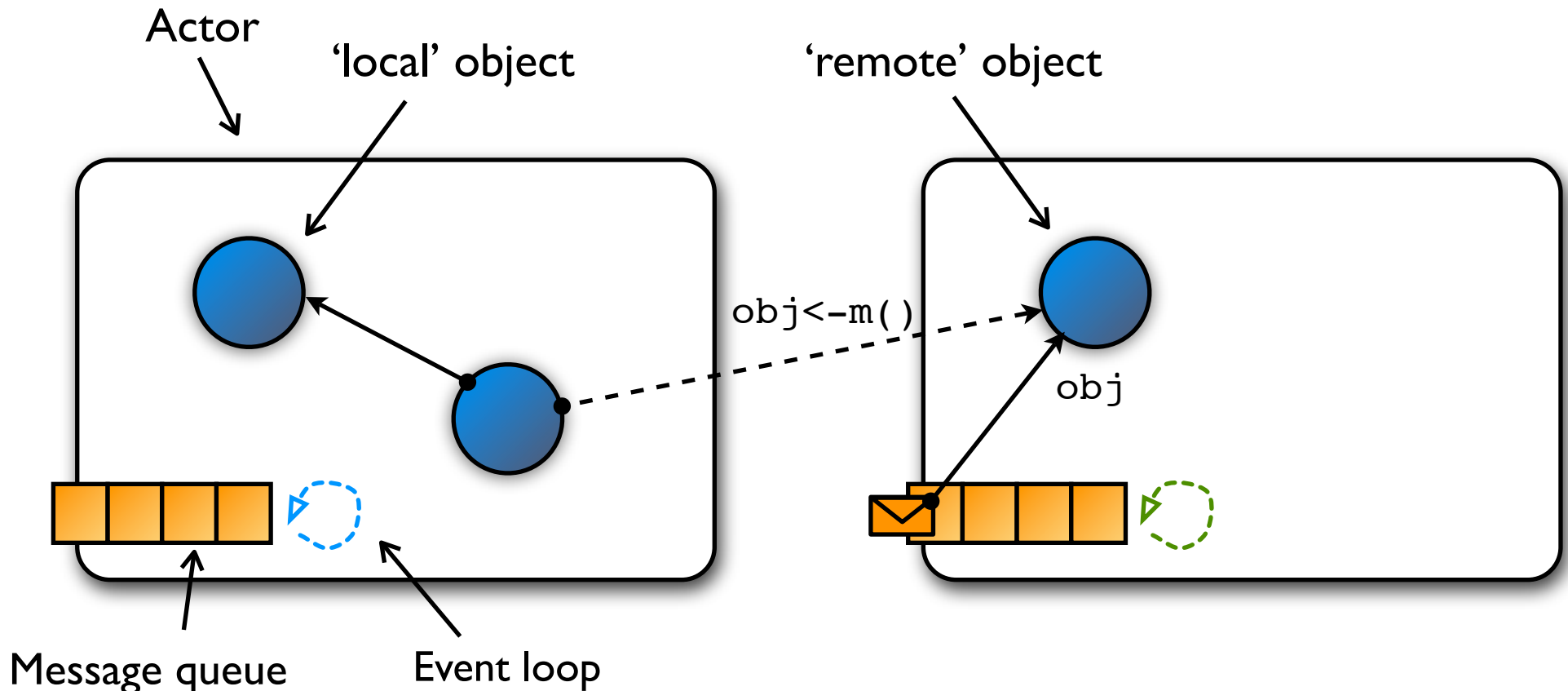
# Event loop concurrency

Based on E programming language [Miller05]



# Event loop concurrency

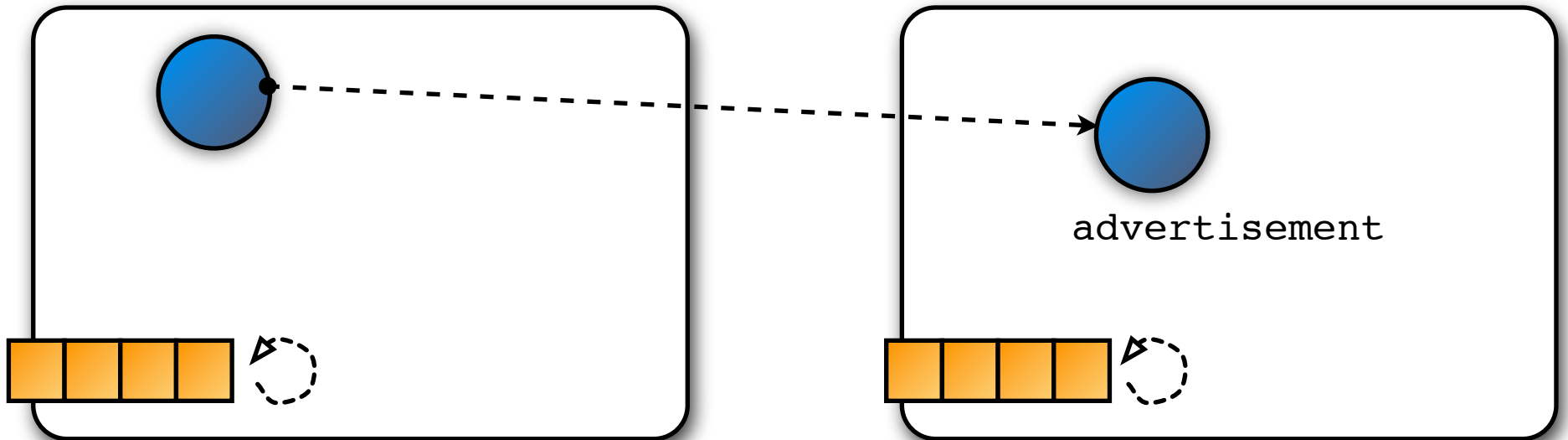
Based on E programming language [Miller05]



Actors cannot cause deadlock  
No race conditions on objects

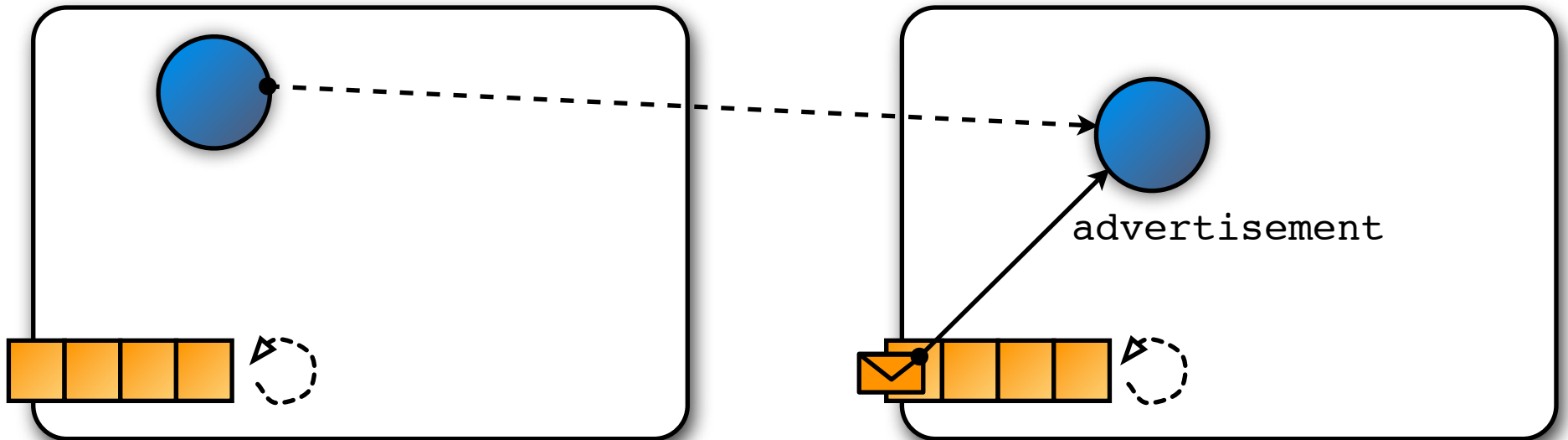
# Futures

```
def future := advertisement<-getContactInfo()
```



# Futures

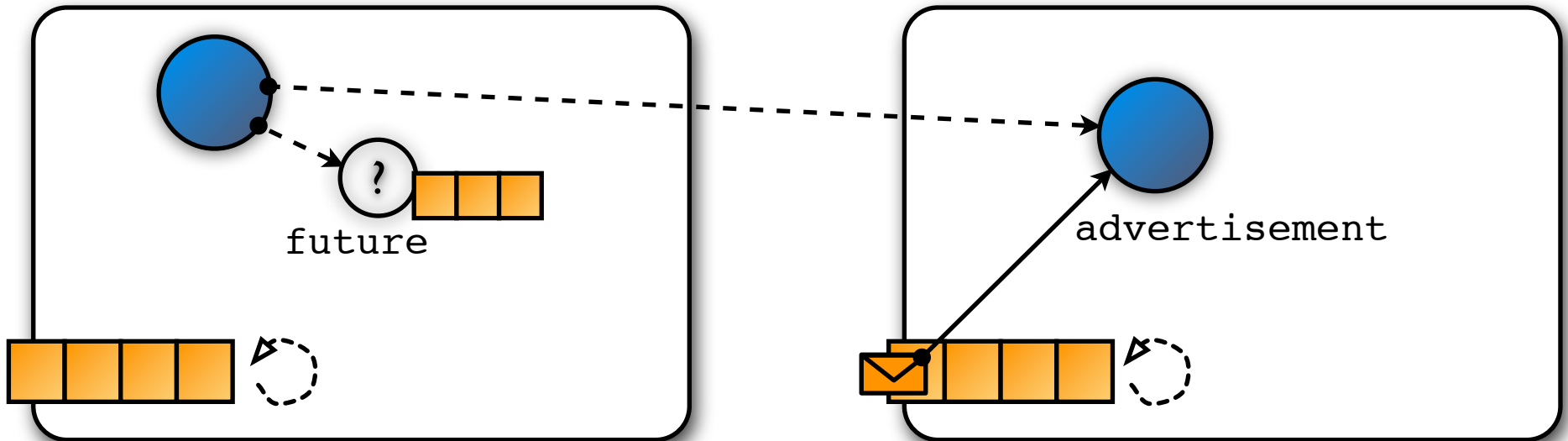
```
def future := advertisement<-getContactInfo()
```





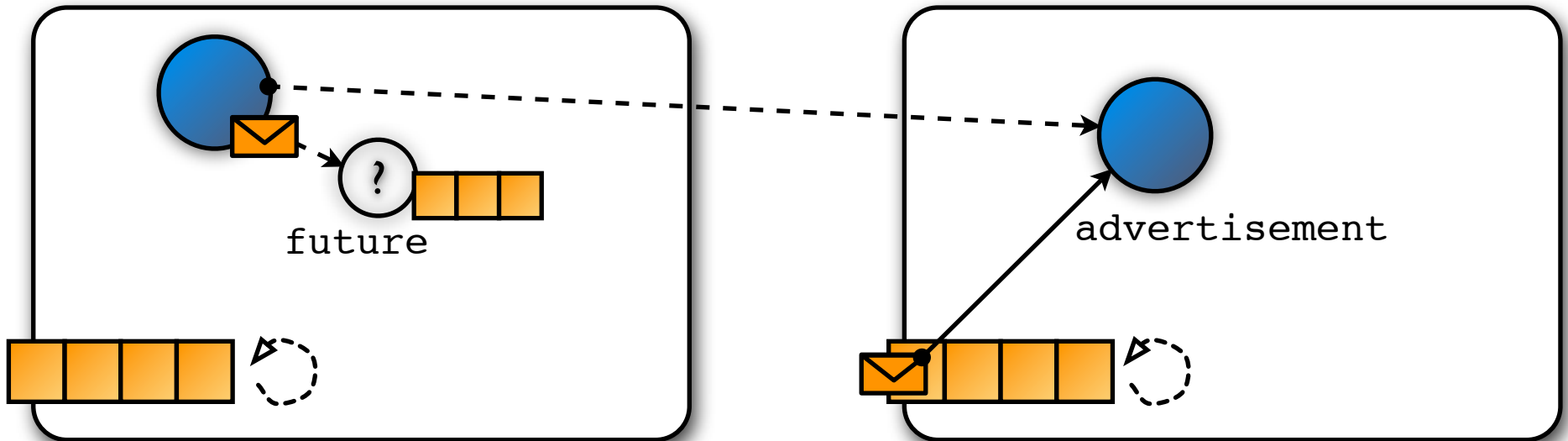
# Futures

```
def future := advertisement<-getContactInfo()
```



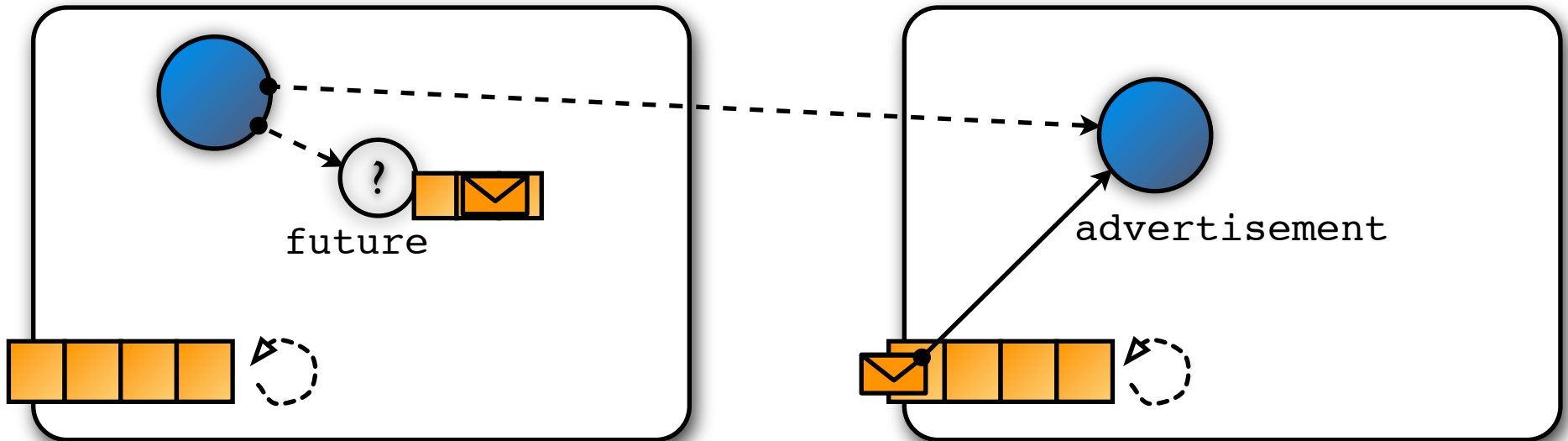
# Futures

```
def future := advertisement<-getContactInfo()
```



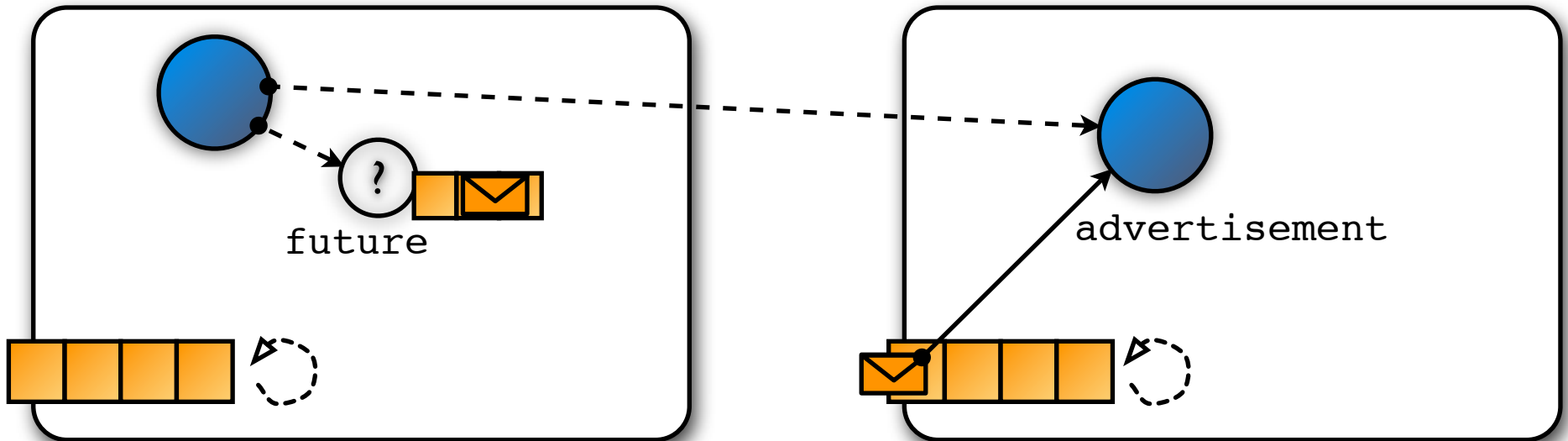
# Futures

```
def future := advertisement<-getContactInfo()
```



# Futures

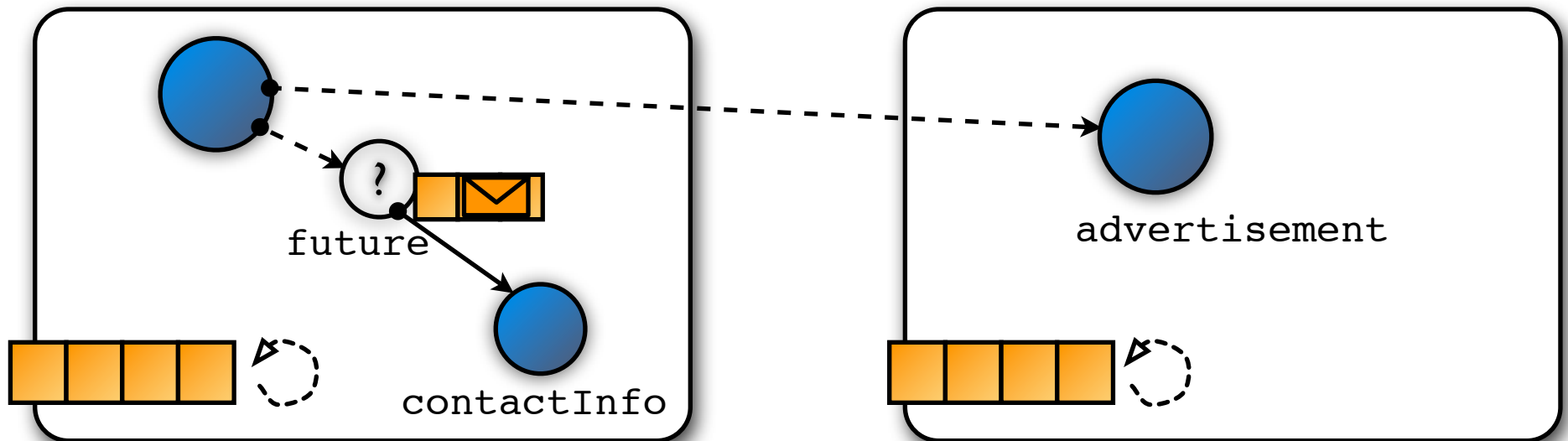
```
def future := advertisement<-getContactInfo()
```



```
when: future becomes: { |contactInfo|  
  println("contact seller: " + contactInfo)  
}
```

# Futures

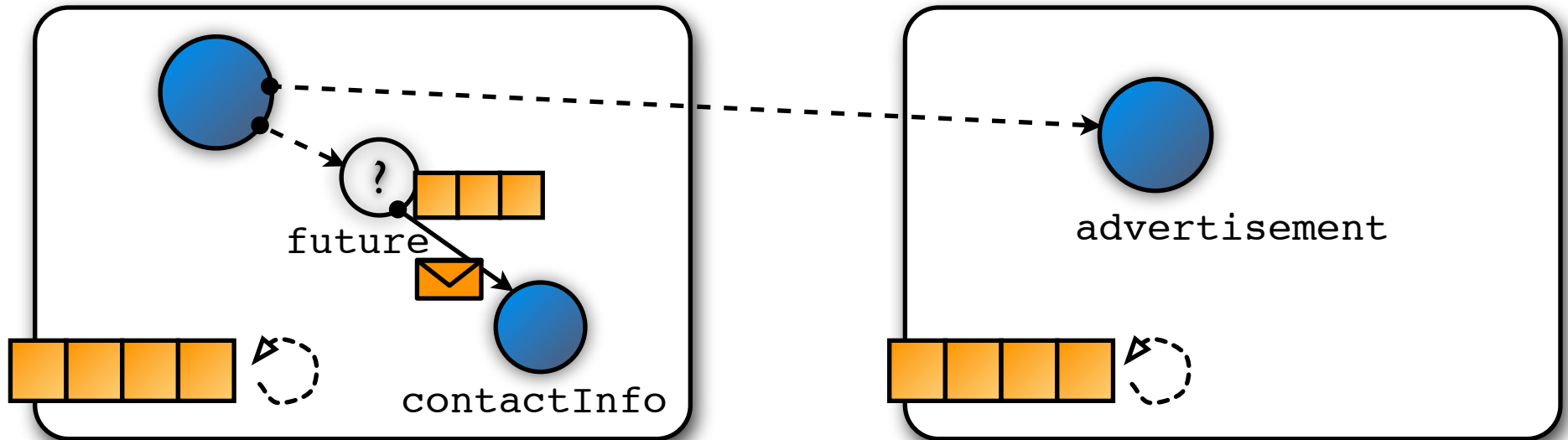
```
def future := advertisement<-getContactInfo()
```



```
when: future becomes: { |contactInfo|  
  println("contact seller: " + contactInfo)  
}
```

# Futures

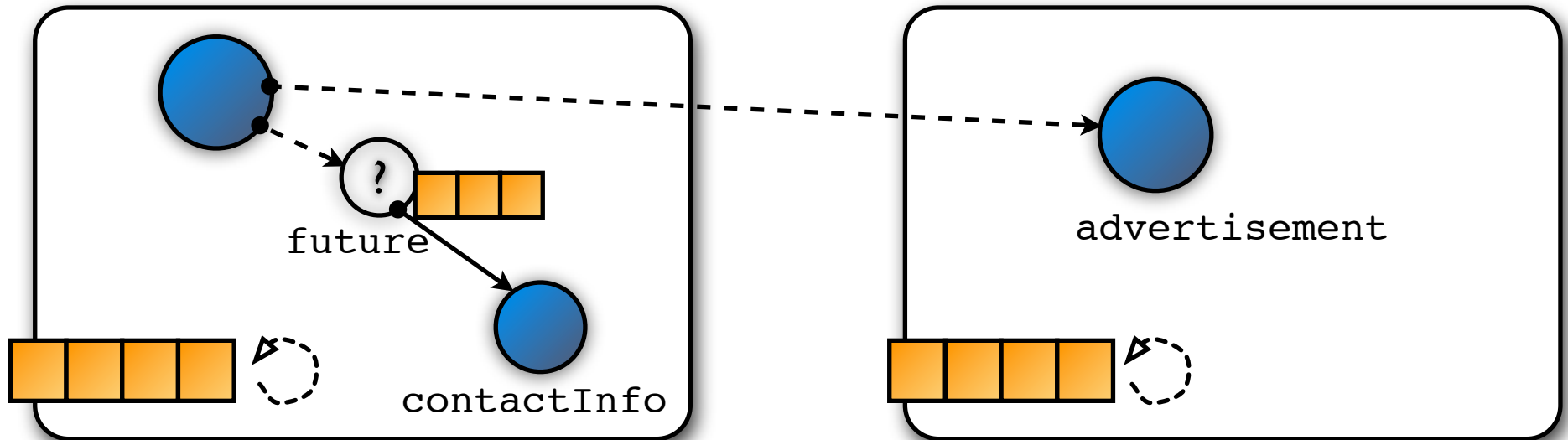
```
def future := advertisement<-getContactInfo()
```



```
when: future becomes: { |contactInfo|  
  println("contact seller: " + contactInfo)  
}
```

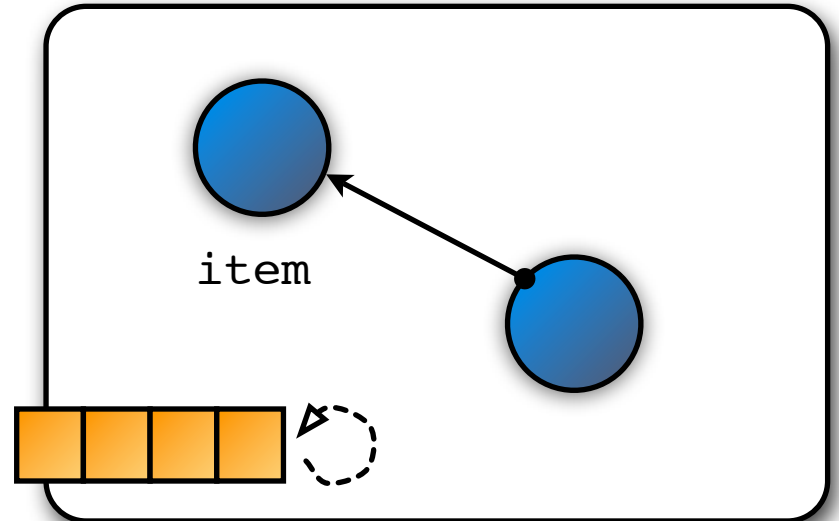
# Futures

```
def future := advertisement<-getContactInfo()
```



```
when: future becomes: { |contactInfo|  
  println("contact seller: " + contactInfo)  
}
```

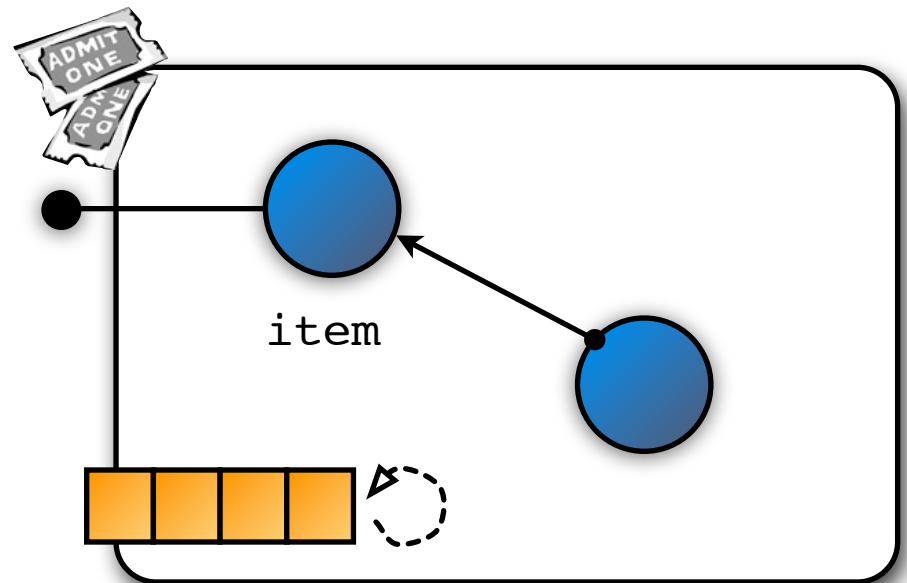
# Exporting objects





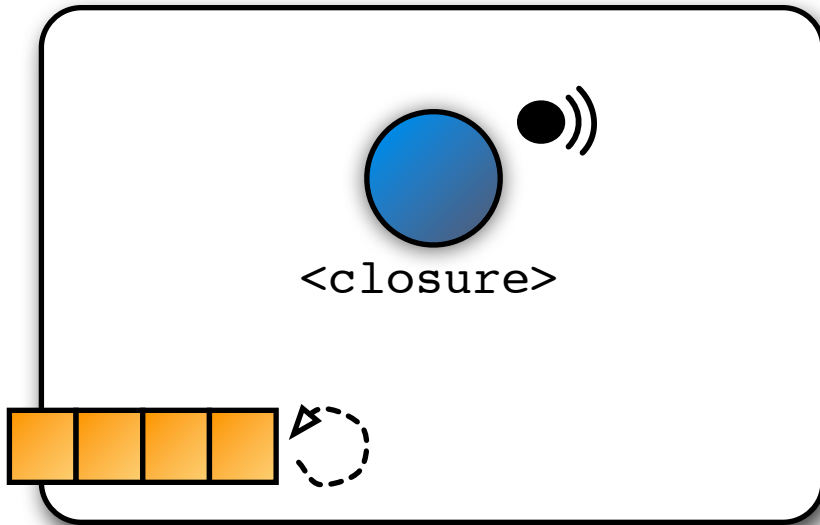
# Exporting objects

```
deftype ConcertTicket;  
  
def Item := object: {  
  def category; // a type tag  
  ...  
  def placeSupply() {  
    export: item as: category;  
  }  
}
```



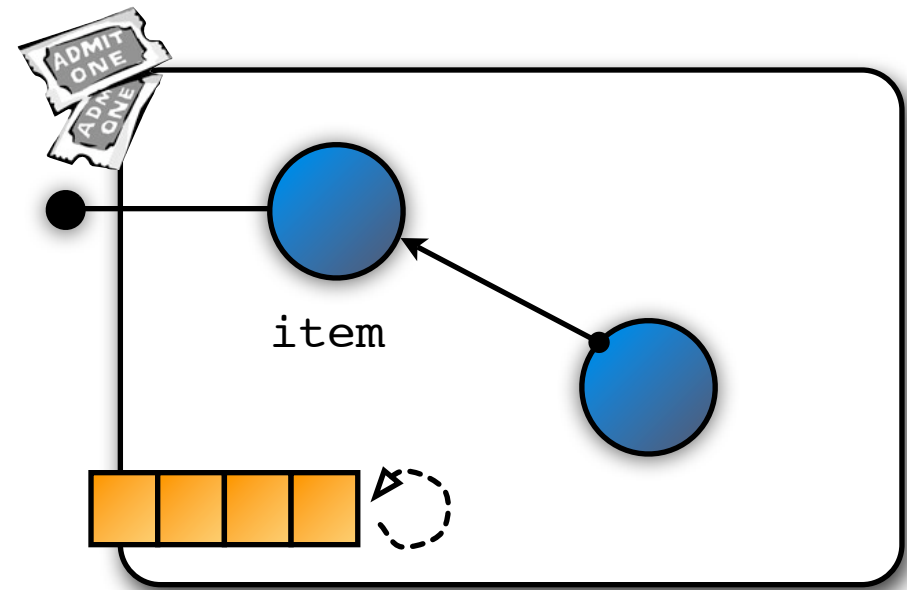
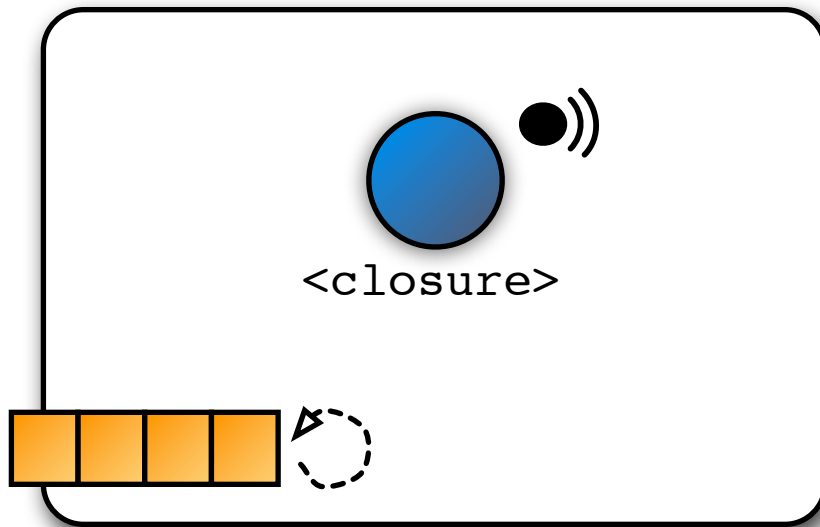
# Service Discovery

```
def placeDemand() {  
  whenever: category discovered: { |item|  
    ...  
  }  
}
```



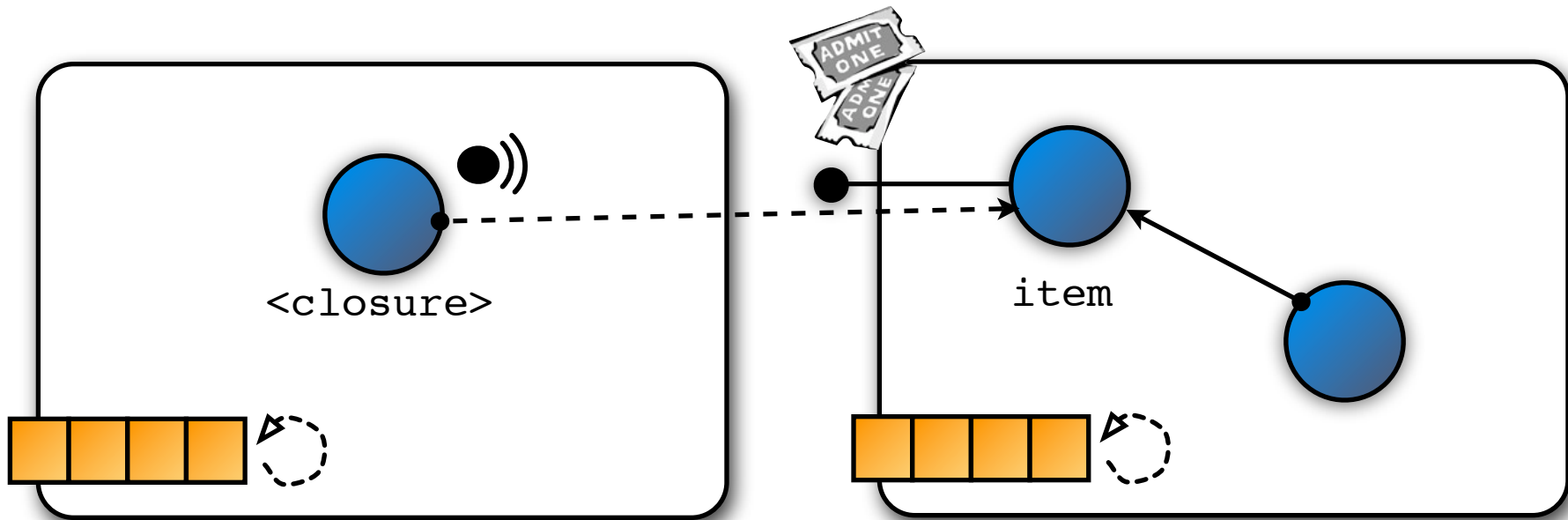
# Service Discovery

```
def placeDemand() {  
  whenever: category discovered: { |item|  
    ...  
  }  
}
```



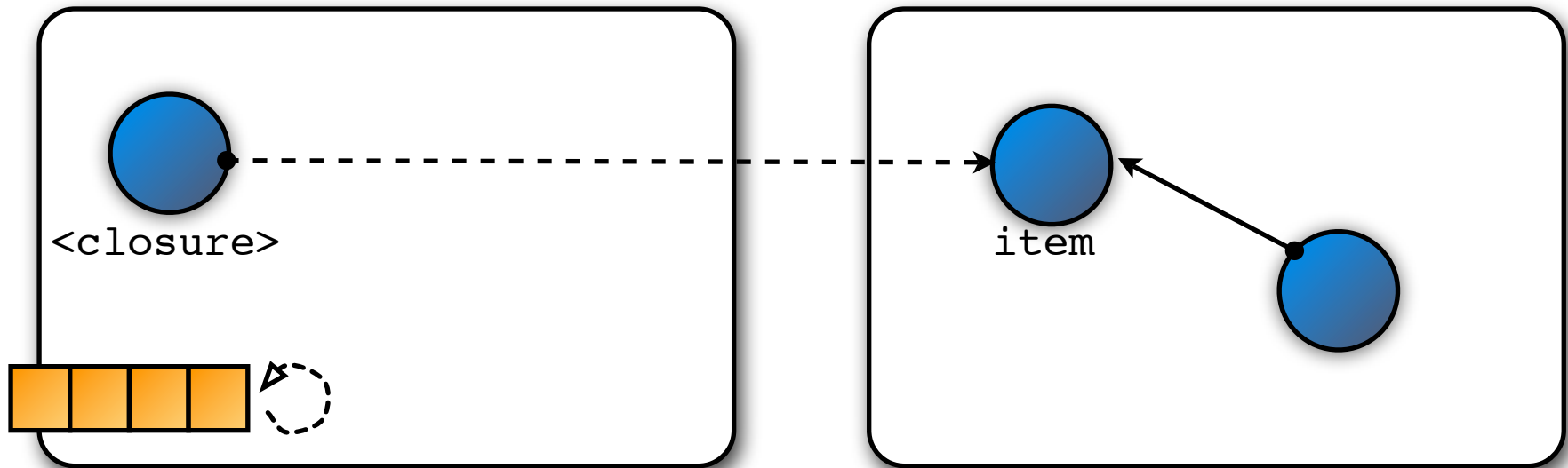
# Service Discovery

```
def placeDemand() {  
  whenever: category discovered: {|item|  
    ...  
  }  
}
```



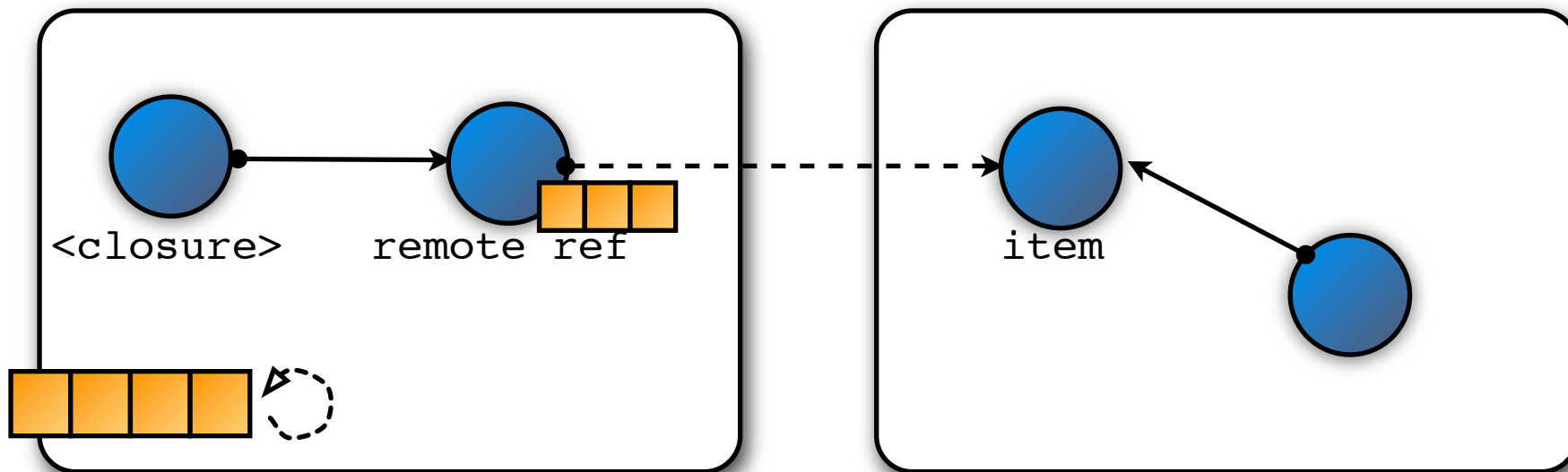
# Failure handling

```
item<-getContactInfo()
```



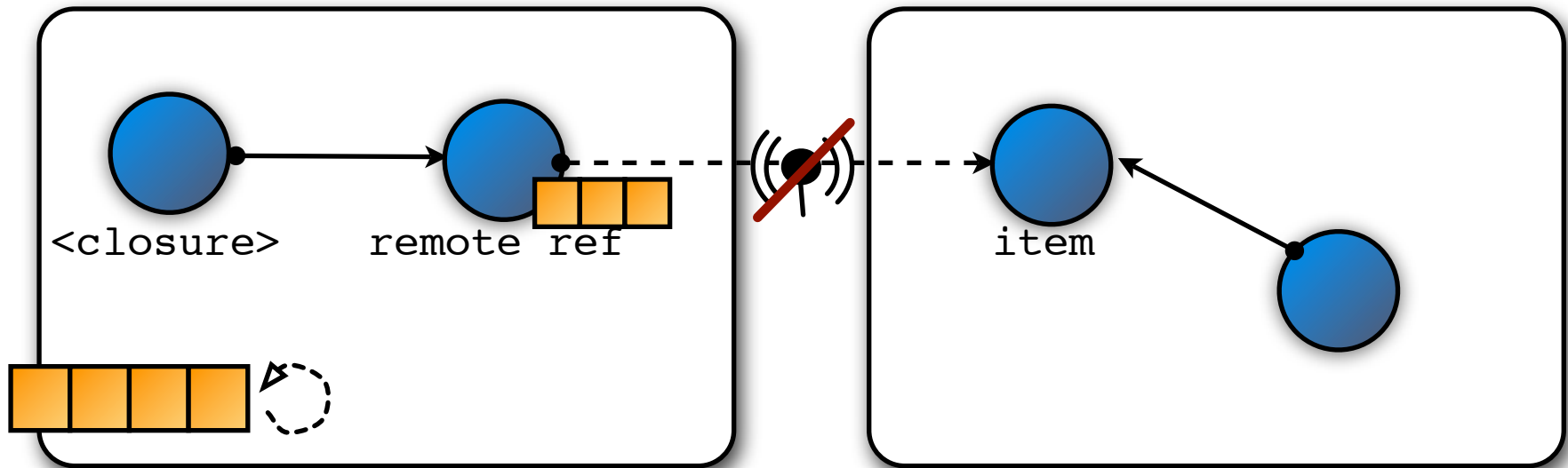
# Failure handling

```
item<-getContactInfo()
```



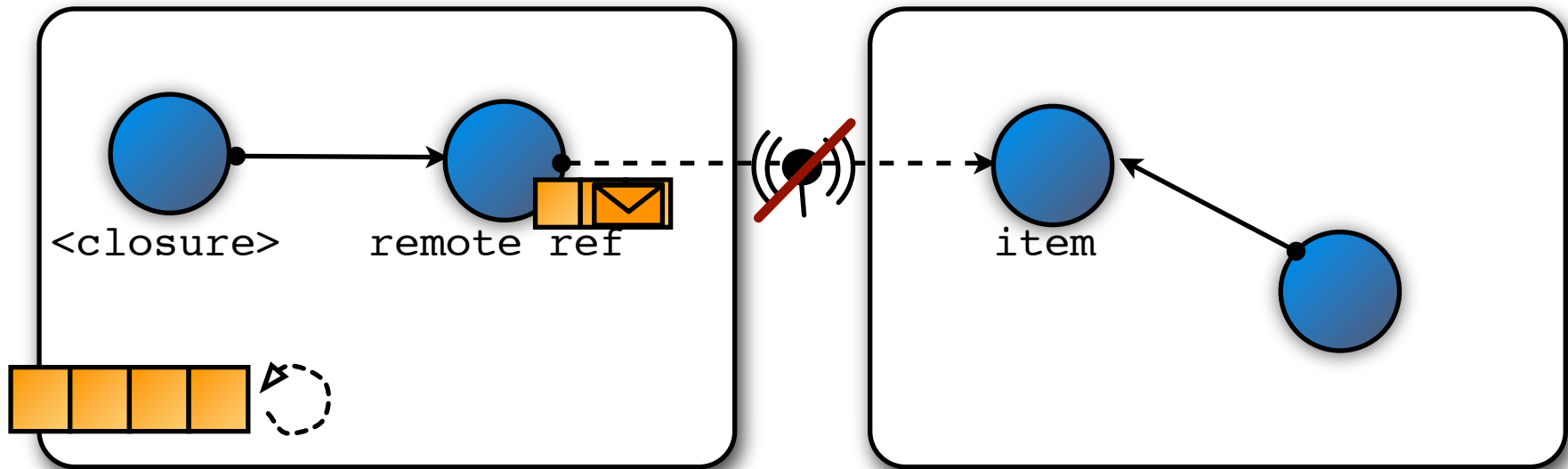
# Failure handling

```
item<-getContactInfo()
```



# Failure handling

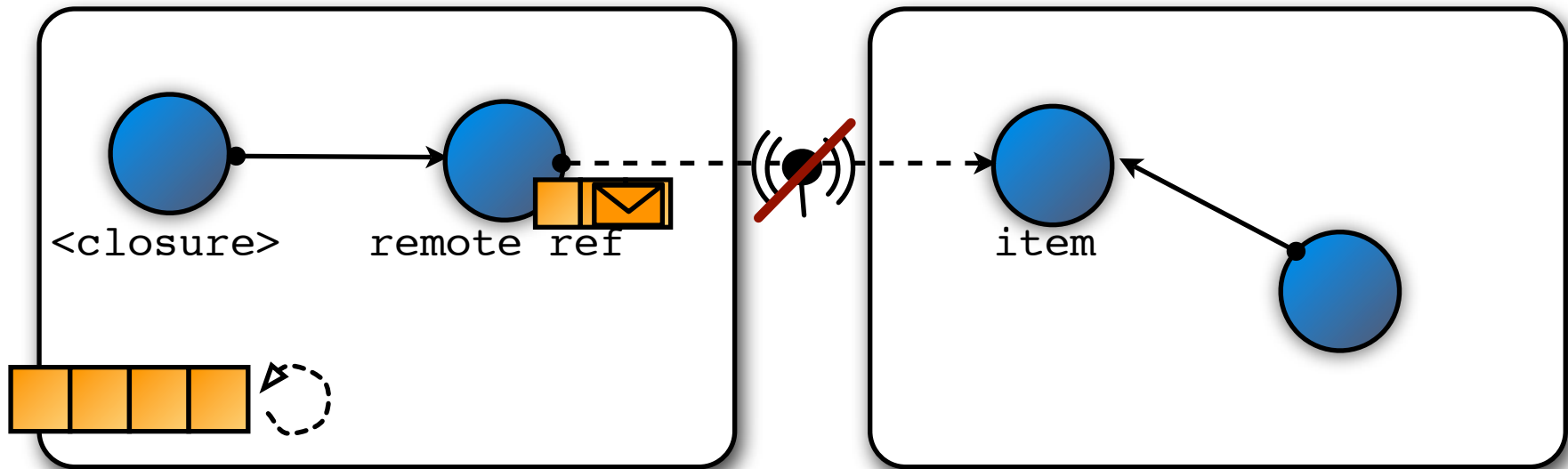
```
item<-getContactInfo()
```





# Failure handling

```
item<-getContactInfo()
```

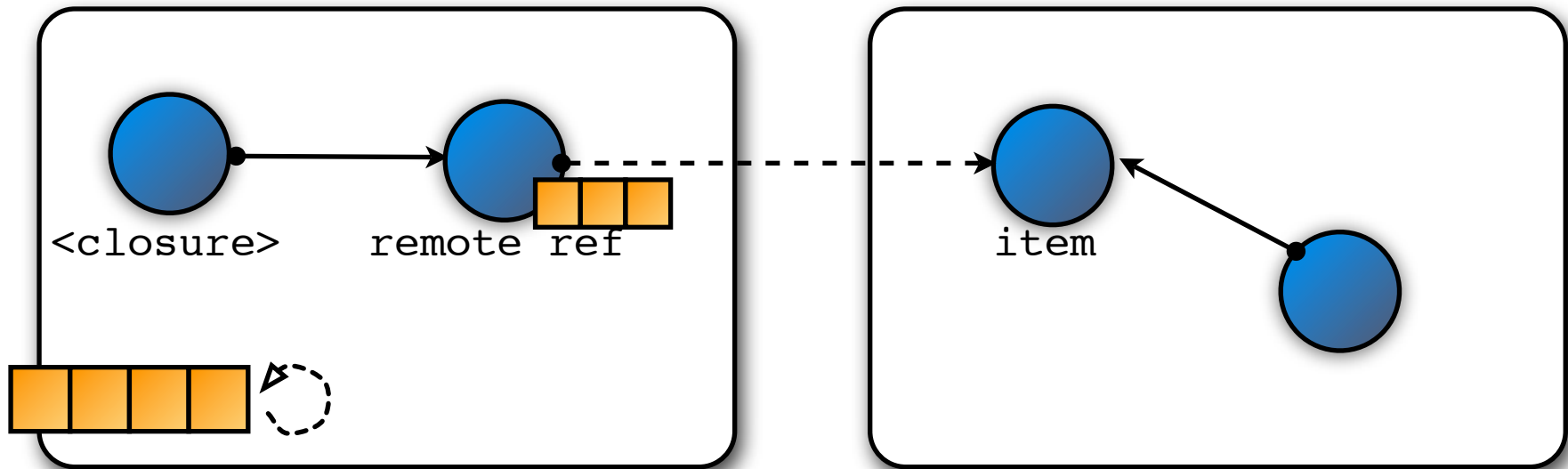


```
when: item disconnected: {  
  println("Item no longer available")  
}
```

```
when: item reconnected: {  
  println("Item available again")  
}
```

# Failure handling

```
item<-getContactInfo()
```



```
when: item disconnected: {  
  println("Item no longer available")  
}
```

```
when: item reconnected: {  
  println("Item available again")  
}
```

# Events + Objects

- Block **closures** as first-class event-handlers
  - preserve state (all lexically visible variables)
  - can be arbitrarily nested
- Leads to less ‘**inversion of control**’

```
whenever: category discovered: { |item|  
  when: item<-getContactInfo() becomes: { |contactInfo|  
    println("contact seller: " + contactInfo)  
  }  
  when: item disconnected: {  
    println("Item no longer available")  
  }  
}
```

# Events + Objects

- Block **closures** as first-class event-handlers
  - preserve state (all lexically visible variables)
  - can be arbitrarily nested
- Leads to less **'inversion of control'**

```
whenever: category discovered: { |item|  
  when: item<-getContactInfo() becomes: { |contactInfo|  
    println("contact seller: " + contactInfo)  
  }  
  when: item disconnected: {  
    println("Item no longer available")  
  }  
}
```

# Conclusion

- MANETs → loosely coupled collaboration
  - (((~~●~~))) Volatile Connections → time & sync-decoupling
    - )) Scarce Infrastructure → space-decoupling
- AmbientTalk: event-driven OO language
  - Buffered **asynchronous messages**: tolerate temporary network failures by default
  - Built-in **service discovery**: no servers required



<http://prog.vub.ac.be/amop>